



Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System

M. Pour, Shahrzad

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
M. Pour, S. (2017). *Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System*.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System

Shahrzad M. Pour

This documentation provides information about the dataset generated as part of a PhD thesis (*Towards Signalling Maintenance Scheduling Problem for European Railway Traffic Management System*) for the signaling maintenance of the Danish railway system. The data instances (M. Pour 2017a; M. Pour 2017b) have been created for the purpose of adaptation to the newest railway signalling standard which is so called European Railway Traffic Management System(ERTMS). The data instances are used particularly in the following research papers:

Clustering of Maintenance Tasks for the Danish Railway System. Published in proceeding of *International Conference on Intelligent Systems Design and Applications*. (Pour and Benlic 2016)

A Choice Function Hyper-heuristic Framework for the Allocation of Maintenance Tasks in Danish Railways. Published in *Journal of Computer & Operations Research*. (M. Pour, Drake, and Burke 2017)

A Constructive Framework to the Preventive signalling Maintenance Crew Scheduling Problem for the Danish Railway systems. Shahrzad M. Pour, Kourosh Marjani Rasmussen, John H. Drake and Edmund K. Burke. Submitted to *Journal of the Operational Research Society*.

The chapter provides explanation of the different types of maintenance tasks in the ERTMS, followed by data definition. Furthermore, it presents information on how the dataset is created and how the software application generates each data file. Data generation is explained through a step by step procedure along with snapshots.

1.1 Signaling maintenance tasks in ERTMS

Signaling maintenance is an essential requirement for the ERTMS implementation. This means that it is necessary to maintain all signaling equipments required for the ERTMS implementation, proportional to any type of railway networks. Overall, there are three different types of signaling equipments for the current ETRMS, including on-board signaling, track-related and pre-installed equipments. Accordingly, three different maintenance tasks are defined, depending on the position of each of these equipments in the system. On-board signaling equipments are the most important components of ERTMS. For installation of such equipments, a complete renewal of the existing system is needed. This can be done by implementation of the European Train Control System (ETCS) which will innately facilitate the maintenance of on-borad equipments through enhancing the accessibility/portability of these devices for the maintenance purpose. This means that the equipments can be transferred directly to the workshop for maintenance which is much easier than the maintenance of the signaling equipment installed along the track, positioned far from the maintenance location. Using this way, we can ensure that the related tasks such as on-site maintenance / inspection of the equipments, installed on the railway tracks, are no longer required.

The second type of tasks contains track-related equipment like balises and point machine for which the crew/engineer is needed for doing the maintenance at the track position. Tasks related to the tracking equipments such as balises and point machines need the crew/engineer support for maintaining the equipments at the the track position. Other than the aforementioned tasks, there are maintenance tasks which need to be handled in the installation points, regardless of whether they are track-related or signaling-related. Examples of such tasks that are required to be performed on-site (in the geographical position of the track) include maintenance of driver screen in the train, the antenna mounted on the top of train and the equipments installed in the radio block center. According to the above categorization, we designed three different set of problems. Each of these sets is different from the other sets according to the location of the maintenance tasks. Each sets of data instances are geographical data indicating tasks and crew locations, task duration, and positioning of the time windows.

1.2 Dataset

Each dataset consists of a set of geographical points, demand, time window constraint, duration and type as below. All the geographical points are located inside the biggest region of Denmark, Jutland. To standardize our dataset, we follow the file format from standard benchmark test-sets for Vehicle Routing Problem Time Windows (VRPTW), introduced by Solomon in 1987 (<http://w.cba.neu.edu/~msolomon/problems.htm>). Since the maintenance planning in Denmark has a decentralized maintenance structure, the crew are located in different locations in the Jutland, meaning that they start their daily tasks from their home location rather than a single depot/station. According to this, the locations of the crew are different from each other in the dataset. We have two sets of rows in each dataset, indicating the number of crew, the number of tasks and their specifications, respectively. The first set of rows is related to the crew which are located in different geographical locations over the region and are distinguished by setting the demand and the duration of the row by zero. In summary, in this set of rows we have the below information:

- Index
- Crew geographical coordination
- Demand = 0,
- time window $[e_0, l_0]$
- Duration = 0,
- Type = 0,

The time window for each crew is used for working hours of the related crew. In this way, we can differentiate between full time and half time crew.

The second set of rows belongs to the maintenance tasks consisting (or including) of the geographical coordination, and the demand of the tasks, used for the synchronization tasks. If a demand of task is one, it means that the task should be done by one crew, if it is two, it means that they should jointly do the task and so on.

- Index
- Maintenance task geographical coordination
- Demand $q_i > 0$,
- Time window $[e_i, l_i]$,
- Duration = 0,
- Type = 0,

Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System

The locations of the crew are identical for all problems, while the set of the maintenance tasks have been randomly generated by utilizing Google Map API in the following categories:

- Random points on whole Jutland area
- points on the railway's lines in Jutland
- Mixed of random points and the exact points on the railways

In order to test the scheduler on different time-horizons, each set of problems has four different numbers of tasks which should be done by a certain number of crews: 100, 500, 1000, and 5000. These numbers are chosen respectively for the number of maintenance tasks needed to be done on daily, weekly, monthly, and half yearly basis according to the current scale of maintenance planning in Denmark. In addition to different maintenance task locations, this helps us to evaluate our approach on clustering the maintenance tasks in different situations when the coordination of the tasks are randomly scattered through the area, are densely located in the railway lines and are a mixture of scattered and on- track points. Figure 1.1 is a snapshot of the text file of one of data instances.

CUSTOMER CUST NO.	XCOORD	YCOORD	DEMAND	READY	TIME	DUE	DATE	DURATION	TYPE
0	55.685200	8.975906	0	0	1236	0	0	0	
1	56.833130	9.017666	0	0	1236	0	0	0	
2	56.274950	8.725060	0	0	1236	0	0	0	
3	56.544780	10.172020	0	0	1236	0	0	0	
4	56.340690	9.482212	0	0	1236	0	0	0	
5	57.227480	10.007057	0	0	1236	0	0	0	
6	56.012130	9.713383	0	0	1236	0	0	0	
7	55.278900	9.168214	0	0	1236	0	0	0	
8	57.473980	9.966450	1	0	1236	1	1	1	
9	57.470500	9.962390	2	0	1236	1	1	1	
10	57.456380	9.985400	1	0	1236	1	1	1	
11	57.538110	9.950130	1	0	1236	1	1	1	
12	56.054720	9.950430	1	0	1236	1	1	1	
13	56.378390	9.891770	1	0	1236	1	1	1	
14	55.648570	9.646910	1	0	1236	1	1	1	
15	55.556710	9.721130	1	0	1236	1	1	1	
16	55.468680	8.792290	1	0	1236	1	1	1	
17	55.471000	9.297240	1	0	1236	1	1	1	
18	55.468710	9.220980	1	0	1236	1	1	1	
19	55.481890	8.999990	1	0	1236	1	1	1	
20	56.369810	10.814060	1	0	1236	1	1	1	
21	56.363140	10.629420	1	0	1236	1	1	1	

Figure 1.1: Snapshot of the text file for one data instance

1.3 Data Generation

We have generated our dataset through three following steps:

1. Finding the Jutland boundary
2. Finding the geographical points on the rail track
3. Generating random points for each dataset

1.3.1 Finding the boundary of Jutland

We have used a drawing application for polyline, polygon, a polygon with holes, rectangle, circle, marker(icon), and direction(route, path). This application uses the Google Maps API Version 3 (V3). It has all the features of Google Maps MyMaps and has direct access to the code for the shapes (overlays). While we drew and created a map of the region (Jutland), KML or Javascript code was presented in the text-area. We copied KML code and pasted it into a text editor. Then we had a KML file including all of the geographical points of the boundary.

Figure 1.2 shows the interface of the Google Maps API v3 Tool and the created boundary of Jutland through this application. For more information, the reader is referred to <http://www.birdtheme.org/useful/v3tool.html>

Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System

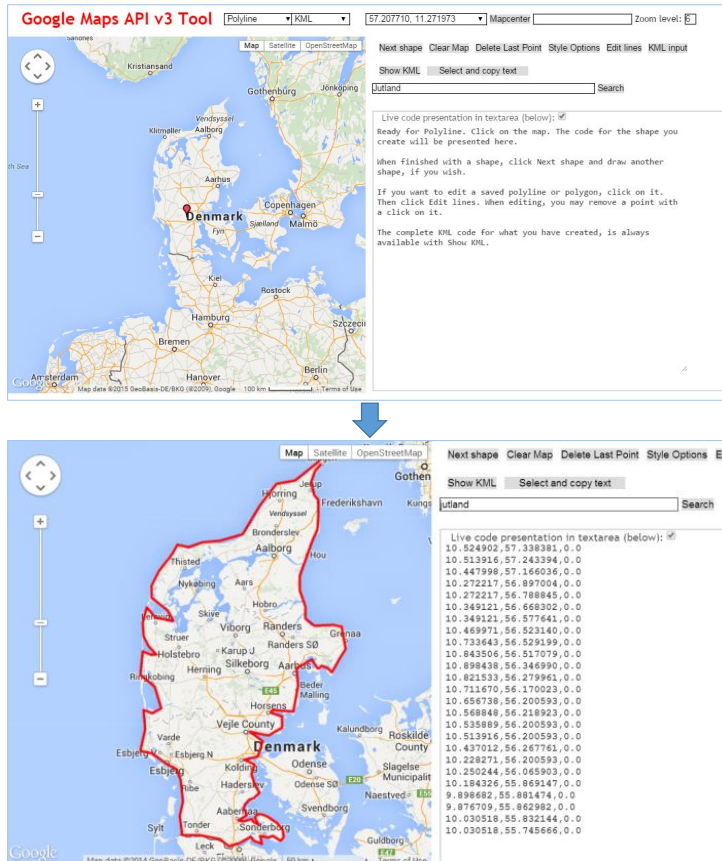


Figure 1.2: *Interface of the Google Maps API v3 Tool and the created boundary of Jutland through this application*

1.3.2 Finding the geographical points on the rail track

In order to generate random points, particularly on the rail track, we prepared a list of routes from different Origins and Destinations. The list covers the whole track routes on the Jutland region. Figure 1.3 shows some of the routes included in the whole set of routes.

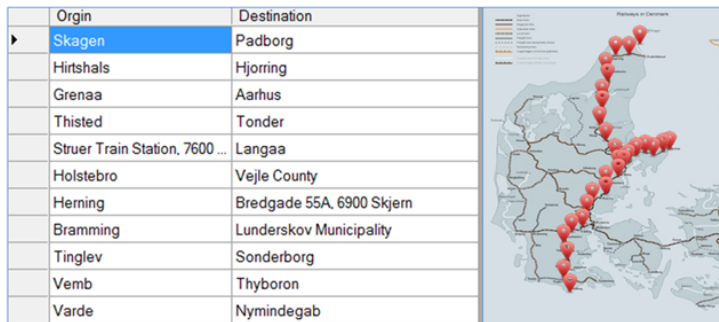


Figure 1.3: *The included routes*

1.3.3 Generating random points for each dataset

For generating random points inside the boundary, we have used the JavaScript (or a JavaScript code) from (shivrajawat 2014). The boundary of the region is given as an input to the script. The script, in turn, generates a number of desired points inside the boundary.

For creating a random set of problems, we have generated a maximum set of random points through the JavaScript (or the aforementioned JavaScript code). Accordingly, we have chosen the number of requested tasks for each problem randomly through a C# random generator function. Similarly, for the problems with tasks on the track, we have chosen the number of tasks required from the collected points on different track routes using the same random function in C#. Figure 1.4 represents the schematic picture of the chosen random tasks after applying the procedure discussed above.

Dataset Documentation for the ERTMS-Oriented Signalling Maintenance in the Danish Railway System

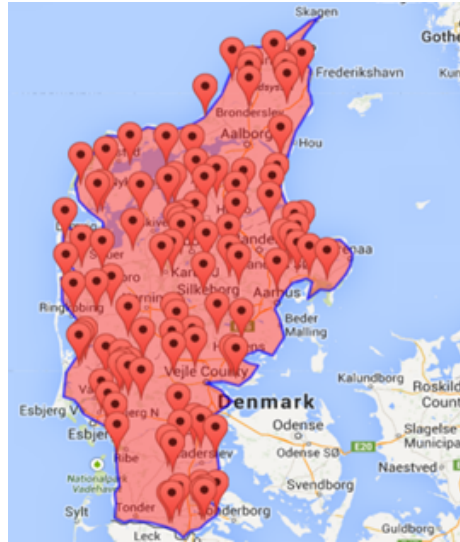


Figure 1.4: *The schematic picture of the chosen random tasks*

1.3.4 Software Application

To generate each data file, we have developed a software using Microsoft Visual Studio C#.Net. Figure 1.5 represents the user interface of our application. The input contains the number of the tasks and the mode of geographical locations. , The output is a text file with the Solomon dataset format. The other parameters in each data have been considered as constant values in the code. For example, the number of the crew has the constant value of 8 in all data instances. However, the software can be updated to get every parameter as input, later on.

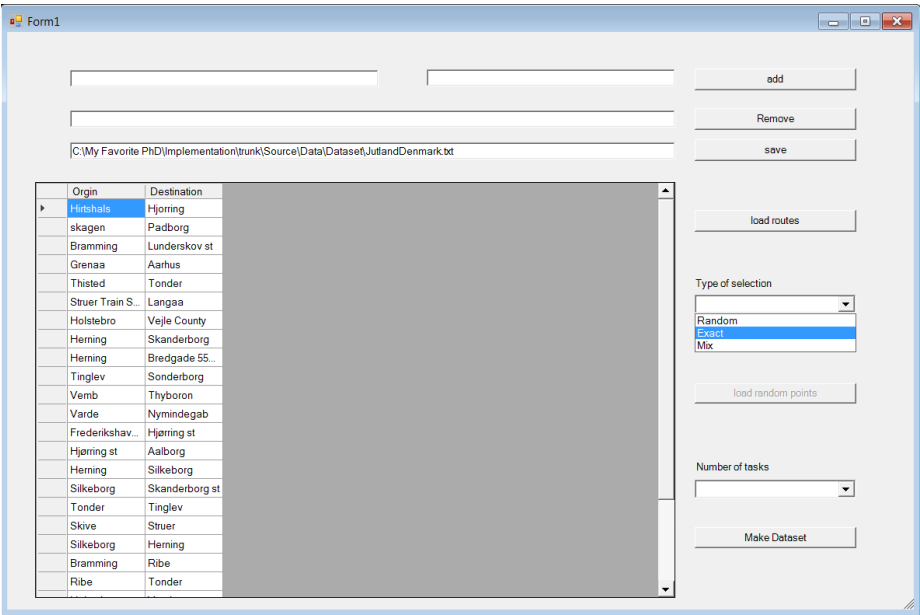


Figure 1.5: The user interface of the application

In order to generate the geographical coordinations exactly on the railway tracks, the software makes use of previously added routes on the rail track of Jutland. However, the software gives possibilities to add additional routes, including more regions into the dataset. Accordingly, to generate the geographical coordination, randomly scattered all over the network, the software loads a pool of generated random points in Jutland by the JavaScript and randomly chooses the number of needed points depending on the size of the dataset. Finally, for generating a mixture of points, the software randomly generates points (By the Random.Next() function in C#) for our two sets of mentioned geographical sources.

1.4 Adopted Java Script code

```
// source: https://github.com/shivrajawat/chicagogit/blob/master/locationselector.php
var map;
var boundaryPolygon;
function initialize() {

    var mapProp = {
```

Dataset Documentation for 10 ERTMS-Oriented Signalling Maintenance in the Danish Railway System

```
        center: new google.maps.LatLng(26.038586842564317,
            75.06787185438634),
        zoom: 6,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };

    map = new google.maps.Map(document.getElementById("map-
        canvas"), mapProp);

    google.maps.Polygon.prototype.Contains = function (
        point) {
        // ray casting algorithm http://rosettacode.org/wiki/Ray-casting\_algorithm
        var crossings = 0,
        path = this.getPath();

        // for each edge
        for (var i = 0; i < path.getLength() ; i++) {
            var a = path.getAt(i),
            j = i + 1;
            if (j >= path.getLength()) {
                j = 0;
            }
            var b = path.getAt(j);
            if (rayCrossesSegment(point, a, b)) {
                crossings++;
            }
        }

        // odd number of crossings?
        return (crossings % 2 == 1);

        function rayCrossesSegment(point, a, b) {
            var px = point.lng(),
            py = point.lat(),
            ax = a.lng(),
            ay = a.lat(),
            bx = b.lng(),
            by = b.lat();
            if (ay > by) {
                ax = b.lng();
                ay = b.lat();
                bx = a.lng();
                by = a.lat();
            }
            if (py == ay || py == by) py += 0.00000001;
            if ((py > by || py < ay) || (px > Math.max(ax,
                bx))) return false;
            if (px < Math.min(ax, bx)) return true;

            var red = (ax != bx) ? ((by - ay) / (bx - ax))
                : Infinity;
            var blue = (ax != px) ? ((py - ay) / (px - ax))
                : Infinity;
            return (blue >= red);
        }
    }
}
```

```

    }
};

google.maps.event.addListener(map, 'click', function (
    event) {
    if (boundaryPolygon != null && boundaryPolygon.
        Contains(event.latLng)) {

        document.getElementById("spnMsg").innerText = "
            This location is " + event.latLng + "
            inside the polygon.";
    } else {
        document.getElementById("spnMsg").innerText = "
            This location is " + event.latLng + "
            outside the polygon.";
    }
});

}

function randomLeftSidepoint(min,max) {
    return (Math.random() * (max - min + 1 ) + min);
}

function randomRightSidepoint(min, max) {
    //var xx = [];
    //xx = random.uniform(min, max).split(".");
    //return xx[1];
    return Math.random() * (max - min + 0.000001) + min;
}

function test() {
    var mingx = 8;
    var mingy = 54;
    var maxgx = 13;
    var maxgy = 57;

    var minlx = 0.033350;
    var minly = 0.010940;
    var maxlx = 0.948807;
    var maxly = 0.983637;

    var points = "";

    var x = [];
    // 1000 is the number of tasks.

    for (var i = 0; i < 1000; i++) {

        var lat = randomLeftSidepoint(mingx, maxgx) +
            randomRightSidepoint(minlx, maxlx);
        var longa = randomLeftSidepoint(mingy, maxgy) +
            randomRightSidepoint(minly, maxly);
        var myLatLng = new google.maps.LatLng(longa, lat);
        while (!boundaryPolygon.Contains(myLatLng)) {

```

Dataset Documentation for 12e ERTMS-Oriented Signalling Maintenance in the Danish Railway System

```
        lat = randomLeftSidepoint(mingx, maxgx) +
              randomRightSidepoint(minlx, maxlx);
        longa = randomLeftSidepoint(mingy, maxgy) +
              randomRightSidepoint(minly, maxly);
        myLatlng = new google.maps.LatLng(longa, lat);
    }
    addpoint(lat, longa);
    points =points+ longa + ' ' + lat + '\r\n';

}
var blob = new Blob([points ], { type: "text/plain;
    charset=utf-8" });
saveAs(blob, "generatepoints.txt");

document.getElementById("spnMsg").innerText = lat+" "
    +longa;

}
function addpoint(lat, longa) {
    var myLatlng = new google.maps.LatLng(longa,lat);
    var marker = new google.maps.Marker({
        position: myLatlng,
        map: map,
        title: ''
    });
}

}

function drawPolygon() {

    initialize();
    // Jutland Boundary
    var boundary = '10.600433 57.742281,10.517178
57.720314,10.431175 57.679276,10.261917
57.610396,10.171795 57.590683,10.076180
57.581274,9.953785 57.581471,9.897308
57.524221,9.826241 57.483308,9.766159
57.436489,9.678955 57.322135,9.516907
57.198608,9.394684 57.151597,9.242935
57.130338,9.085693 57.129947,8.979950
57.144266,8.794556 57.088532,8.682632
57.095344,8.589935 57.096187,8.442993
56.973898,8.342743 56.900827,8.289185
56.826265,8.331070 56.735041,8.427887
56.676953,8.555603 56.612296,8.560753
56.553361,8.510971 56.524696,8.378448
56.561212,8.182068 56.591816,8.175201
56.444204,8.342056 56.294402,8.161812
56.304913,8.179321 56.165969,8.323517
56.051575,8.423767 55.924909,8.296051
55.843596,8.206787 55.762283,8.189621
55.629687,8.279572 55.608629,8.329353
55.577584,8.368149 55.527906,8.536377
55.470536,8.608303 55.444748,8.674736
```

```

55.387797,8.676624 55.297220,8.689499
55.205069,8.709755 55.115360,8.667698
55.072085,8.680573 55.013083,8.667870
54.923993,8.757648 54.903097,8.819962
54.916936,8.996773 54.898843,9.157104
54.879169,9.237270 54.858106,9.317436
54.814907,9.384384 54.848705,9.475536
54.861229,9.539223 54.899042,9.632864
54.945540,9.660587 54.982573,9.570208
55.024316,9.463348 55.010940,9.513302
55.084769,9.469872 55.153863,9.657669
55.210399,9.632263 55.309465,9.605999
55.374576,9.585228 55.438124,9.635997
55.475603,9.546025 55.480333,9.467039
55.497514,9.623508 55.528894,9.733200
55.579236,9.785299 55.604378,9.727535
55.629520,9.675179 55.665850,9.532013
55.707529,9.688396 55.716817,9.844780
55.688967,9.948807 55.738502,10.030861
55.806534,9.940052 55.822031,9.846497
55.852958,10.033350 55.892624,10.157032
55.862967,10.272560 55.965395,10.230160
56.072406,10.182266 56.127278,10.226383
56.178310,10.286980 56.223227,10.351696
56.274229,10.427399 56.288624,10.482416
56.297731,10.531940 56.279400,10.488167
56.181695,10.616055 56.238167,10.696478
56.229759,10.743942 56.242724,10.854149
56.321904,10.902386 56.372136,10.917664
56.439075,10.872688 56.475219,10.802994
56.515889,10.685749 56.513132,10.579491
56.492187,10.400448 56.514446,10.276337
56.606379,10.326118 56.662977,10.244064
56.795008,10.236168 56.893823,10.299683
56.983637,10.377960 57.119815,10.511169
57.244063,10.531082 57.266087,10.524559
57.309762,10.501556 57.341558,10.503616
57.388821,10.527649 57.458987,10.475464
57.505496,10.437012 57.534239,10.409546
57.576252,10.434952 57.618493,10.476837
57.659263,10.601807 57.743747';

var boundarydata = new Array();

var latlongs = boundary.split(",");

for (var i = 0; i < latlongs.length; i++) {
    latlong = latlongs[i].trim().split(" ");
    boundarydata[i] = new google.maps.LatLng(latlong
        [1], latlong[0]);
}

boundaryPolygon = new google.maps.Polygon({
    path: boundarydata,

```

```
        strokeColor: "#0000FF",
        strokeOpacity: 0.8,
        strokeWeight: 2,
        fillColor: 'Red',
        fillOpacity: 0.4

    });

    google.maps.event.addListener(boundaryPolygon, 'click',
        function (event) {
            document.getElementById("spnMsg").innerText = '';
            if (boundaryPolygon.Contains(event.latLng)) {
                document.getElementById("spnMsg").innerText = "
                This location is " + event.latLng + "
                inside the polygon.";
            } else {
                document.getElementById("spnMsg").innerText = "
                This location is " + event.latLng + "
                outside the polygon.";
            }
        });

    map.setZoom(5);
    map.setCenter(boundarydata[0]);
    boundaryPolygon.setMap(map);
}
```

References

- M. Pour, Shahrzad (2017a). *Jutland Dataset with Centralized Crew/Depot Location*. <http://github.com/ShahrzadMP/RegionSplitterDataset>.
- M. Pour, Shahrzad (2017b). *Jutland Dataset with Random Crew/Depot Location*. <http://github.com/ShahrzadMP/Dataset>.
- M. Pour, Shahrzad, John H Drake, and Edmund K Burke (2017). "A choice function hyper-heuristic framework for the allocation of maintenance tasks in Danish railways". In: *Computers & Operations Research*.
- Pour, Shahrzad M and Una Benlic (2016). "Clustering of Maintenance Tasks for the Danish Railway System". In: *International Conference on Intelligent Systems Design and Applications*. Springer, pages 791–799.
- shivrajawat (2014). *locationselector*. <https://github.com/shivrajawat/chicagogit/blob/master/locationselector.php>.